AD-A053 021 GENERAL RESEARCH CORP SANTA BARBARA CALIF F/G 9/2
COST REPORTING ELEMENTS AND ACTIVITY COST TRADE-OFFS FOR DEFENS--ETC(U)
MAY 77 C A GRAVER, W M CARRIERE F19628-76-C-0180
UNCLASSIFIED CR-1-721-VOL-2 ESD-TR-77-262-VOL-2 NL

1 OF 1
AD A053021

END
DATE FILMED
6-78
DDC

ESD-TR-77-262, Vol. II

COST REPORTING ELEMENTS AND ACTIVITY COST TRADEOFFS FOR DEFENSE SYSTEM SOFTWARE (EXECUTIVE SUMMARY)

General Research Corporation
P. O. Box 3587
Santa Barbara, CA 93105

May 1977

Approved for Public Release;
Distribution Unlimited.

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
HANSCOM AIR FORCE BASE, MA 01731

LEGAL NOTICE

When U.S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

This technical report has been reviewed and is approved for publication.

William J. White

WILLIAM J. WHITE, Captain, USAF
Project Engineer

John T. Holland

JOHN T. HOLLAND, Lt Col, USAF
Chief, Techniques Engineering Division

FOR THE COMMANDER

Toru Yamamoto

TORU YAMAMOTO, Colonel, USAF
Director, Computer Systems Engineering
Deputy for Command & Management Systems

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER (18) ESD-TR-77-262, Vol. II (2)	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) (6) Cost Reporting Elements and Activity Cost Trade-offs for Defense System Software, Volume II, Executive Summary	5. TYPE OF REPORT & PERIOD COVERED (9) Final Report, 11 Mar 76 - 11 Mar 77
	6. PERFORMING ORG. REPORT NUMBER (14) CR-1-721-Vol-II
7. AUTHOR(s) (10) C.A. Graver, W.M. Carriere, E.E. Balkovich, R. Thibodeau	8. CONTRACT OR GRANT NUMBER(s) (15) F19628-76-C-0180
9. PERFORMING ORGANIZATION NAME AND ADDRESS General Research Corporation P.O. Box 3587 Santa Barbara, CA 93105	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (16) PE63101F, Project E234
11. CONTROLLING OFFICE NAME AND ADDRESS Electronic Systems Division Air Force Systems Command Hanscom AFB, MA 01731	12. REPORT DATE (11) May 77 13. NUMBER OF PAGES 56 (12) 53p.
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same	15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

ESD Project Engineer: Captain William J. White (MCI)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software	Software Life Cycle	Software Size
Costs	Software Development	Computer Program Components
Reporting System	Software Maintenance	
Software Costs	Software Manhour Estimates	
Software Reporting System	Software Product Description	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This technical report examines the costs of developing and maintaining computer software for major defense systems. A process model is described which depicts the relations among activites and phases of the software life cycle, identifies the product and cost information that is normally available, and specifies the milestones. This process model is normally used as the basis for selecting the elements of a software cost reporting system. The suggested reporting system also includes descriptions of the final product, time phasing of product development, a standardized list of Computer Program Components, and a standardized

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

402 754

BB

(Block 20 continued)

list of labor categories.

During the study, data was collected from several sources including the following Air Force organizations;

Electronic Systems Division
Aeronautical Systems Division
Space and Missile Systems Organization, and
Data Systems Design Center

Cost estimating relationships for each phase of the software life cycle are explored, using the process model and the data. The importance of trade-offs in cost between phases is demonstrated. The report also contains estimating relationships for evaluating the cost effects of software size, computer capacity constraints, programming language, and changes in requirements. It also addresses the separation of two activities, error correction and product improvement, during the maintenance phase of the life cycle. Results are integrated with other software cost estimating techniques.

ABSTRACT

This technical report examines the costs of developing and maintaining computer software for major defense systems. A process model is described which depicts the relations among activities and phases of the software life cycle, identifies the product and cost information that is normally available, and specifies the milestones. This process model is used as the basis for selecting the elements of a software cost reporting system. The suggested reporting system also includes descriptions of the final product, time phasing of product development, a standardized list of Computer Program Components, and a standardized list of labor categories.

During the study, data was collected from several sources including the following Air Force organizations:

Electronic Systems Division
Aeronautical Systems Division
Space and Missile Systems Organization
Data Systems Design Center

Cost estimating relationships for each phase of the software life cycle are explored, using the process model and the data. The importance of trade-offs in cost between phases is demonstrated. The report also contains estimating relationships for evaluating the cost effects of software size, computer capacity constraints, programming language, and changes in requirements. It also addresses the separation of two activities, error correction and product improvement, during the maintenance phase of the life cycle.

Results are reported in Vol. I and summarized in this volume. Section 1 is a 17-page summary of the entire project. Sections 2 and 3 highlight the reporting-system and estimating-relationship results respectively.

CONTENTS

TABLES

1 SUMMARY

1.1 GOALS

In April 1976, General Research Corporation (GRC) began a study of "Life-Cycle Costing of Major Defense System Software and Computer Resources," Contract F19628-76-C-0180. The purpose was to assist Air Force Program Offices and staff agencies in estimating, reporting, and controlling the cost of the computer software embedded in defense systems. The study was performed under the direction of the Electronic Systems Division (AFSC), Computer Systems Engineering Office (TOI). Captain William White was the Project Officer and coordinator of the data-collection survey, and has provided many helpful comments during the course of this study.

The study had two goals:

1. To define the elements of a system for reporting the costs of developing and maintaining computer software
2. To develop resource estimating relationships for the different phases of the software life cycle

The reporting system has two purposes: (1) collection of consistent and accurate data, to be used for estimating the costs of future software, and (2) provision of timely and relevant information concerning the progress of software development, to be used by Program Offices in cost control. It is truly unfortunate that the need for a reporting system still exists. As early as 1966, reporting systems were being devised for collecting data on software cost.[1] So far, however, no uniform reporting system has been adopted by the Air Force or DoD.

The second goal, development of resource estimating relationships for the different phases of the software life cycle, has also had a long history of attempts. In general, previous studies concentrated on estimating total cost. Their lack of success was not due to lack of competence; many imaginative relationships were developed and tested.

Unfortunately, variances remained too large for such relationships to be useful for estimation, as witnessed by the fact that authors did not recommend the use of their findings to estimate software cost.

In an attempt to attack the problem from a different perspective, we were directed to develop resource estimating relationships (that is, equations for man-hours, computer-hours, and elapsed time) for each phase of the software life cycle separately, rather than for total man-hours or costs. The relationships were to be based upon currently available data and developed using parametric cost estimating techniques.

The following principles evolved during the course of the study and guided our progress towards the study's goals.

First, both the recommended cost reporting system and the estimating relationships must be related to the Air Force procurement process. Attempts now underway to standardize this process are reported in AFR 800-14,[2] which we have used as our guide in developing a Process Model, tempered by visits to Program Offices and by our own experience.

Second, the cost reporting system must meet two objectives. It must provide information which will help the Program Offices to control software costs. At the same time it must be used to compile data on many systems, using standard definitions, so that better cost estimating relationships can eventually be developed.

Third, resource estimating relationships for man-hours, computer-hours, and elapsed time should be developed separately for each phase of the software life cycle, using parametric cost estimating techniques. The relationships should estimate resources as functions of inputs normally available to the Air Force. A relationship which depends on input data not available to the Program Office (for example, competence of the individual programmer) might be statistically valid but would be useless in practice.

The first goal of the study has been achieved. The elements of a reporting system have been defined. Elements to describe product status as well as resource consumption are included.

The development of estimating relationships was less successful. New estimating relationships were developed, but they do not constitute a complete model of software cost, nor a complete set of resource estimating relationships for each phase of the life cycle. They can only be viewed as first steps in developing such a model.

The reasons for this partial achievement are described in Sec. 1.2, Obstacles. Our conclusions are presented in Sec. 1.3, and recommendations in Sec. 1.4. Section 1 constitutes a complete overview of the study.

More details concerning the reporting system and the estimating relationships are given in Secs. 2 and 3, respectively.

Volume 1 also includes (in Sec. 7) a description of the state of the art in software cost estimating techniques, which incorporates the results of our study with those of others reported in the literature. The techniques still have wide variances, and caution should be used in applying them. This portion of Vol. 1 is not summarized in this volume.

1.2 OBSTACLES

The obstacles we met included definitional problems, data problems, and conceptual problems. Each is discussed below.

1.2.1 Definitional Problems

One of our first tasks was to develop a model of the software life cycle. The basis for the model was AFR 800-14,[2] which divides the software life cycle into six phases: (1) analysis, (2) design, (3) coding and checkout, (4) test and integration, (5) installation, and (6) operations and support. The first four phases are generally referred to as

the development period. The process is a complicated one, including feedback and overlapping among the phases.

A further complication is the dual use of the terms for life-cycle phases. For example, "Design" refers to a set of activities: functional allocation of the software to modules, development of the detailed design as represented by flow diagrams, etc. On the other hand, "Design" is also commonly described as beginning after the Preliminary Design Review (PDR) and ending at the Critical Design Review (CDR). Thus, both an "activity" and a "milestone" use of the term are common.

This ambiguity caused considerable problems during the study. One consequence was inconsistencies in the data. Some sources reported data on an activity basis, but most sources derived their reports on resource consumption from time-phased expenditure records which can, at best, be associated with milestones.

Although we feel that the dual use of these terms should be eliminated, the problem is so widespread that redefining the terms in this study would probably have simply caused more confusion. We have chosen, therefore, to specifically identify the problem and to show how the two uses are related. (A mapping of one use onto the other is contained in Table 2.3). In addition, we define the dual uses of the terms in Sec. 2 of Vol. 1 and adopt the following convention: "When the terms analysis, design, and so on are used without qualifiers, an activity definition is intended. When the terms are followed by phase or milestone, a milestone definition is intended."

1.2.2 Data Problems

As in previous studies of software cost, the location and collection of accurate and consistent data was a problem. We first attempted to collect data from seven representative Program Offices in the Air Force Systems Command: three at the Electronic Systems Division (ESD),

two at the Aeronautical Systems Division (ASD), and two at the Space and Missile Systems Organization (SAMSO).

The results of our visits to these Program Offices were extremely important in shaping the study's direction, especially in defining the elements of the recommended reporting system. The Program Offices have a difficult problem in controlling the cost of software. A major reason is that costs are reported at far too aggregated a level--in some cases as only a single total. The problem is complicated by the inability to relate software cost to progress towards the completion of software development. Progress is often reported by the percentage of estimated manhours expended to date; a less than reliable measure of the amount of software actually completed. Also, since there is no standard list of software products (end items) for which costs have been collected in previous projects, the Program Offices have few precedents upon which to base cost estimates.

Recorded cost data exhibits a great deal of variability between developments. This is due not only to the size and complexity of the developments, but also to the following:

1. Fixed-price contracts have gone to ceiling. Hence the costs reported understate the costs incurred.
2. Cost-reimbursement contracts have been augmented by supplementary agreements that incorporate changes in requirements.* This has resulted in redoing work already completed, so that costs are abnormally high per line of code delivered.
3. Level-of-effort maintenance contracts include costs for product improvement as well as error correction. Costs are difficult to assign to these two different functions.

*Generally authenticated by Engineering Change Proposals (ECPs).

4. Software costs are often hidden in hardware development due to the difficulty in some instances of allocating the costs between hardware and software (e.g., in systems-engineering functions).

For these reasons, we concluded that the Program Offices were not a good source of data on the resource requirements for individual life-cycle phases, within the study's time and cost constraints. The quality of the data was poor because of the lack of uniform reporting, as described above. Also, there was not sufficient detail relating the degree of product completion to cost. Finally, what data existed was difficult to collect and did not include all the life-cycle phases, especially maintenance. At some point the data from the Program Offices should be compiled into a usable data base, but the effort required will be well beyond the resources of this contract.

At the first Technical Direction meeting, we proposed to take the following study approach. Data would be collected from readily available sources in which data on software product development could be measured. Of special concern was visibility into both development and maintenance portions of the life cycle. SAMSO contractors appeared to be a good source because they had both development and maintenance responsibility. In particular, we gathered data from the SAMSO Program Office responsible for the development of general-purpose support software for the Satellite Control Facility (SCF).

A major SCF subsystem is the Advanced Orbital Ephemeris Subsystem (AOES), which contains more than four hundred programs including compilers, operating systems, data reduction and presentation utilities, and orbit planning and analysis utilities. The AOES is maintained by two associated contractors and an integration contractor. To achieve configuration control, the integration contractor maintains a data base of the characteristics of the AOES programs, and records describing their modifications.

Unfortunately, this data cannot be correlated with the costs of the component activities. Also data on the size of the programs is missing. However, it still was possible to use this data to investigate the properties of the operations and support phase of the software life cycle.

The data we collected describes four successive revisions or "releases" of the AOES, and the maintenance histories of those releases once they became operational. For each program of each release, we collected (1) the size as delivered, (2) the number of problems reported and resolved, (3) the size of the Part II Specifications, (4) the design changes incorporated with that release, and (5) the relationship of system integration tests to the design changes. In addition, we collected data on the man-months of development and the man-months of maintenance effort, and the schedules for developing, testing, and operating the releases.

A third data source was located late in the study at the Air Force Data Systems Design Center (AFDSDC). Software development projects at the Design Center are supported by an automated system called the Planning and Resource Management Information System (PARMIS). The system is a project planning and management aid that enables project managers to enter estimated schedules and manpower allocations and to receive regular reports on actual utilization.

The histories of over 2,000 software developments are stored on the system. PARMIS files its data by activity; a project manager entering a new project into the system estimates start and end times and manhours for each activity, selected from a standard set of activities contained in the PARMIS Catalog. The Catalog's activity codes and activity group codes make it possible to allocate the costs of the development activities among the phases.

PARMIS records planned and actual man-hours, start dates, and end dates (for each activity) under four job classifications: Functional

Analyst, Data Systems Analyst, Programmer, and Support Personnel. The project manager is free to revise his estimated dates and man-hours, but the system always maintains and reports the estimates made when the project was entered into the system.

PARMIS has two drawbacks. First, it contains little information on the nature of the software being developed. We had to obtain information on the size and functions of programs by personal interview. Although people at the Design Center were very cooperative, the interviews made data collection a lengthy process. Second, PARMIS contains only business-application computer programs (payroll, administration, etc.). We do not consider this a serious limitation, however, as the development process is similar to that of major defense-system software, and therefore a great deal of information can be gained from studying the development process at AFDSDC.

We also used some data on our own software developments, and data from previous studies reported in the open literature. However, most of this data was too aggregated to be used in phase-specific studies.

In short, collection of software cost data continues to be a problem (highlighting the need for a good reporting system). Of the data bases investigated, PARMIS was the most complete, containing records of expenditures related to software development activities and phases. In addition, the AFDSDC retains responsibility for software maintenance, and therefore also collects data on errors. Our study represented only a limited attempt to tap this data base, examining only 20 out of 2,000 recorded developments. Of these 20, we were only successful in obtaining complete data on seven, due to incomplete questionnaires. Thus, the limited use of PARMIS in this study should be considered a beginning of its investigation and not an end.

1.2.3 Conceptual Problems

The last major obstacle concerned the attempt to develop resource estimates for each of the life-cycle phases separately. While we were collecting data, we hypothesized resource estimating relationships and causal parameters for each of the life-cycle phases. Section 3 of Vol. I documents the results.

In trying to calibrate these relationships with the PARMIS data, it became very clear that the phases are not independent. That is, the resource consumption during Test and Integration depends on the amount of time spent on Analysis and Design; the number of errors found during Maintenance is dependent upon the amount and quality of Development Testing; and so on.

Although conceptually a model incorporating these trade-offs is more difficult, it does offer the opportunity to provide more accurate estimating relationships and to discover optimal resource allocations among phases. Figure 1.1 shows such a trade-off relationship for five PARMIS data points. Although five points is far too small a sample to offer conclusive results (and other relationships could fit the data equally well), the possibility of discovering these types of trade-offs is exciting, and PARMIS offers a good vehicle for the attempt.

These trade-offs may hold the key for determining why aggregated estimating techniques have such wide variance. Data used to derive these aggregated relationships included less costly developments, with efficient resource allocations among the phases, along with more costly developments with inefficient allocations. Also, discovering efficient allocations will give the Program Offices insight into the planning of software development. Preliminary investigations along these lines are described in Secs. 4 and 5 of Vol. I.

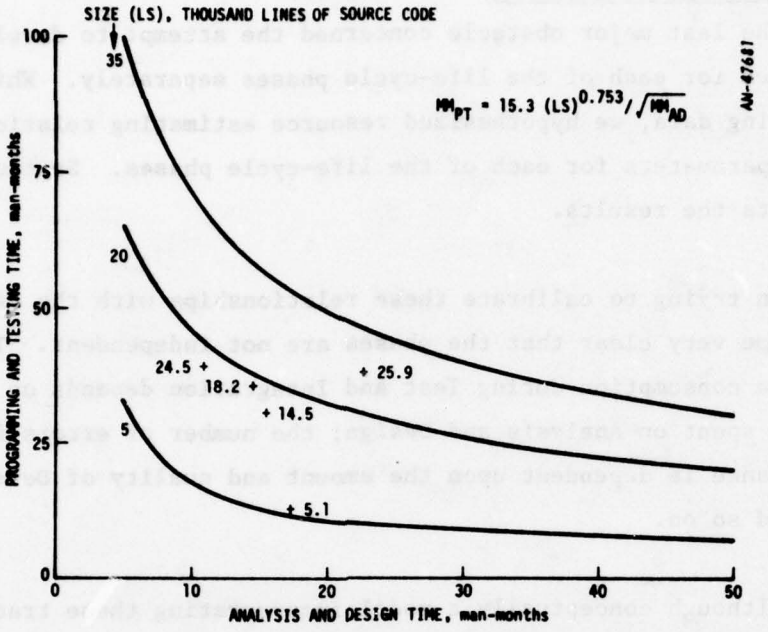

Figure 1.1. Relationship Between Analysis and Design Time and Programming and Testing Time (Volume I, Fig. 5.10)

1.3 CONCLUSIONS

Although these obstacles caused considerable problems in meeting the objectives of the study, we did complete the selection of elements for the cost reporting system and develop some new resource estimating relationships. Significant findings are reported in this section, organized around the two study goals.

1.3.1 Reporting-System Elements

Before summarizing our findings, we briefly describe the philosophy used to pick the specific elements and categories of data to be reported. One approach would be to define the data that is wanted and develop the reporting system necessary to collect it, assuming that the data is there for the asking and that once defined, all that is required is the money to finance its collection. In effect, many of the previously proposed

data collection systems take this point of view. An important question to resolve is why they were not implemented.

One possible explanation is that the cost of collecting data for these systems was simply too high. However, we feel that an equally significant reason is contractors' unwillingness to provide data in as much detail as the proposed systems called for. In effect, what a contractor is technically able and willing to report internally for management control is far more than what he is willing to share with the government. Excessive requirements, therefore, will meet with stiff resistance from contractors or even discourage them from continuing to provide the services; neither of which is desirable.

In selecting the data to be collected, we have tried to be sensitive to this problem. In general, we have recommended more detail in reports that are only one-time requirements. We have recommended less in regularly required reports, but still sufficient for control by Program Offices and the development of estimating relationships.

Section 8 of Vol. I presents our recommendations for resource-data elements. They call for the regular collection of data on costs, man-hours, computer-hours, and the degree of product completion. Other information to be collected at specified milestones includes a detailed description of the software product, the characteristics of the computer system used to develop and/or maintain the software, etc. This information is much more detailed than that required to be regularly reported; however, the infrequency of reporting should ease the burden on the contractor.

A summary of the reporting system elements is contained in Sec. 2 of this volume. Important innovations include the following:

1. A moving milestone (source-code baseline) providing a continuous measurement of progress towards completion. This milestone separates Coding and Checkout from Test and Integration. Each time a source deck is completed (coded and debugged), the deck is "baselined" and future work on that part of the program will be recorded against Test and Integration. Thus, work in these two phases will go on concurrently and data will be continuously available on the percentage of the product which has finished the initial coding and debugging phase and is in the Test and Integration phase.

2. No requirement to separate redesign from recoding efforts in the Test and Integration phase. We speculate that this requirement in earlier cost reporting systems has been one of the reasons they were not implemented. Redesign and recoding efforts in this period are often done by the same person and therefore difficult to separate. If separate reporting were required, it would probably be artificial, so the information content would be suspect.

3. A preliminary standardized list of end items (CPCs*) for comparing resource consumption among software developments. This list will serve to disaggregate the software product into meaningful parts. Ultimately, software will be described by its parts, with estimates made at the parts level and then aggregated, much as hardware estimates are made today. The list is preliminary in that we are confident that additions will be made upon review by the software community.

4. Separation of maintenance into error response and product improvement. Only the need for error response should be associated with the original software development project. Product improvement (through ECPs) should undergo separate

*Computer Program Components (CPCs) are in general a disaggregation of Computer Program Configuration Items (CPCIs).

approval and be subject to data gathering as another development project.

5. Identification of differences in "milestone" and "activity" definitions for Analysis, Design, Coding, and Testing. The inconsistent use of these terms has been a source of confusion during the entire effort, as previously discussed, and has undoubtedly contributed to confusion in other attempts at data gathering and analysis. Ultimately, different terms for the milestone and activity uses should be selected.

In addition to defining the data elements, we have demonstrated one way of mapping these elements into an already existing Air Force reporting system (Program Breakdown Codes--AFSCM 173-4).[3] Hence, the mechanics of data retrieval will not be severe.

We believe these innovations will improve the chances that contractors will accept a reporting system, without sacrificing the needs of the Program Offices for cost control data and the long-term needs of the Air Force for better data to improve cost estimation.

1.3.2 Software Cost Estimating Relationships

Our second major objective was to develop improved software cost estimating relationships (CERs). A significant amount of work had been previously devoted to this task by competent groups, focusing on estimating total man-hours or costs. Results had been disappointing, with derived relationships exhibiting large variances.

As a result, we were directed to take a new approach and focus our attention on the development of separate estimating relationships for each life-cycle phase. Our efforts have been concentrated on man-hour estimation.

During the contract we first developed a process model to describe the relations between activities and phases and to identify data sources normally available during the life cycle. This model is described in detail in Sec. 2 of Vol. I. We then hypothesized estimating relationships for each of the life-cycle phases.

An important factor in all the estimating relationships was the size of the software. Various measures of size have been used in earlier studies, and there has been a great deal of debate on whether a measure of source code or object code is the most appropriate. In estimating cost separately for each phase of the life cycle, much of this debate can be sidestepped by choosing object code as the measure in some phases and source code in others. Our choice is to use object code as the measure in all phases except Coding and Checkout. In that phase, source code is the product being manufactured and seems to us the most appropriate measure of the effort.

Beyond this choice, there is further debate on what to count in measuring the size of software; specifically, whether to include:

Throwaway code--code written and used for the development of the delivered programs, but not itself delivered

Converted code--code taken from earlier programs but adapted in detail for this development

Off-the-shelf code--code taken from earlier programs without adaptation

Computer data--code that supplies values for variables in the program (such as DATA statements in FORTRAN programs)

Comment statements--code intended to be read by people and not by the computer

Our choices are as follows. For all phases but Coding and Checkout, the measure of size is the number of object-code instructions delivered,

excluding only computer data (and of course throwaway code, which is not delivered). In the Coding and Checkout phase, our recommended measure is the number of source-code instructions delivered, again excluding throwaway code and computer data, and also excluding off-the-shelf code. These choices are explained in Sec. 3 of Vol. I.

We then attempted to fit the PARMIS data to the hypothesized relationships. Important findings include:

1. Accurate estimating relationships for each life-cycle phase cannot be developed independent of the other phases. The data when plotted exhibit significant variation. As a consequence, an estimate of total man-hours by estimating each life-cycle phase and then adding will have less precision than an estimate made on the total.

2. Estimating the trade-offs between the life-cycle phases is of prime importance. Only then will the variation in the total man-hour estimates be reduced, i.e., the accuracy of the estimating relationship be increased. In effect, covariances must be incorporated in the estimators. Reducing the risk in software cost estimates is dependent on quantifying these relations between the life-cycle phases.

3. Estimating the trade-offs can also lead to the development of rules for optimal allocation of resources among life-cycle phases. Application of the rules, in turn, will lead to less costly or lower-risk developments. In effect, these rules will represent estimating relationships for software developments with an efficient allocation of resources. Note that we are implying that much of the variation in previous estimating techniques is the result of mixing in the same data base those software developments which have inefficient with those which have efficient allocations of man-hours among life-cycle phases.

We also developed some new aggregated estimating relationships. These are summarized in Sec. 3 of this volume and include:

1. The relationships of aggregate software man-hours to key driving variables such as size, language, capacity constraints, and requirement changes (ECPs) during development.

2. The relationships among different measures of the size of software such as number of lines of source code, size of Part II Specifications, and number of object-code instructions. Conversion ratios between object and source code for different languages and machines have been developed.

3. The magnitude of maintenance activities and the separation of error correction from product improvement. Examples are given in which product improvement has been the major consumer of resources during maintenance. Relationships for estimating error rate were examined, and the difference in maintenance manning requirements for source and object programs was quantified.

Detailed derivations and measures of the quality of the relationships are contained in Sec. 6 of Vol. I.

1.4 RECOMMENDATIONS FOR FUTURE WORK

On the basis of this study, we believe that the following efforts will best contribute to the complicated job of estimating the cost of software.

First, and of utmost importance, implement a cost reporting system. If previous systems had been implemented, we would have had a usable cost data base today. We hope our suggested elements are the basis for the approved cost reporting system, as we believe they avoid some of the impediments to acceptance by contractors, and offer a great deal of control for the Program Offices. However, the important thing is to implement some system. Furthermore, we recommend that the use of the reporting system be a contractual requirement.

Secondly, we recommend that the Air Force continue the study of the PARMIS data base begun in this contract. PARMIS is the only readily available data base in which the relations (trade-offs) between phases can be examined. Although our results to date are quantitatively inconclusive, because of the small sample extracted from the data base,* the existence of the relations has been demonstrated. Because the programs described in PARMIS are uniform as regards important cost-driving parameters such as computer language and type of applications, and because PARMIS and the AFDSDC organization make it possible to obtain very detailed information on the programs, we believe that further study of the PARMIS data base affords a good opportunity to quantify the relations among phases.

The development process for business-application software at AFDSDC is similar to that for major defense system software. Therefore, the functional forms of the relationships between the costs of life-cycle phases, if not the relationships themselves (with some scaling rules), can be used for defense system applications. It is most important to quantify these relationships because they are the key to identifying proper resource allocations among the life-cycle phases and thereby achieving risk reduction in software developments.

Thirdly and concurrently, a systematic collection of data from current and completed projects sponsored by Program Offices should be initiated. Although this is expensive and time-consuming, we cannot afford to wait 10 to 15 years until the data from a reporting system (implemented today on new software projects) is usable for cost estimation.

*2000 projects are included in the data base, from which we gathered data on 20. Most of the data base remains to be examined.

2 REPORTING SYSTEM ELEMENTS

The first goal of the study was to develop a recommended set of elements for a software reporting system around which a uniform data collection system can be designed. We were asked to identify only the elements of a system. The implementation of the system--forms, formats, information flows, etc.--was beyond the scope of our study.

Four primary goals guided our thinking in selecting the reporting system elements. The system should:

1. Provide for cost control by Program Offices. The major short-run benefit of the reporting system will be the visibility given to the Program Offices for controlling software costs.

2. Provide a data base for better cost estimates. The long-run benefit of the reporting system will be the development of a software cost data base with consistent definitions. The necessity of such a data base for developing accurate Cost Estimating Relationships is evident.

3. Have a reasonable chance of wide acceptance by contractors. Perhaps earlier data collection systems[1] were not implemented because they asked for too much detail. There must be a trade-off between (1) the detailed data desirable for supporting cost estimation, (2) the more aggregated information suitable for cost control, and (3) the even more aggregated information that contractors will readily provide. The wider the difference between (1) and (3), the more likely that contractors will resist providing the information.

4. Be based on measurable items. Detailed costs should be gathered around typical contractor work packages and not artificial groupings of costs. Furthermore, these work packages should be standardized so that comparisons can be made between programs.

Using these goals, we developed the following three principles which have guided our selection of elements:

1. Resource consumption should be related to progress towards software product completion. This is the single most important item for cost control.

2. The level of detail should not be greater than specifically needed for cost control and the prediction of future resource consumption.

3. Data requested should relate to the actual development process and not require artificial allocations.

The reporting system elements were defined using these principles as a guide. Conceptually, these elements have three dimensions: the software end items being developed, the resources being consumed, and the phase of the software life-cycle in which the resource consumption takes place. All three dimensions are important for cost control and cost estimation. Also of great importance is to collect information on the software product's degree of completion along with the resources consumed. Only then will the relationship between resources and product be established.

This summary presentation of the reporting system elements will address the following topics: Life-Cycle Phases, Product Status, Software End Items, Reporting System Elements, and the feasibility of quick implementation by using an existing reporting system (AFSCM 173-4).[3] For a more complete development, the reader is referred to Sec. 8 of Vol. I.

2.1 LIFE-CYCLE PHASES

The following "milestone" definitions are adopted for the life-cycle phases:

Phase	Beginning Milestone
Analysis	Contract Award
Design	Preliminary Design Review (PDR)
Coding and Checkout	Critical Design Review (CDR)
Internal Test and Integration	Source-Deck Baseline
Qualification Test	First Preliminary Qualification Test (PQT)
Installation	Physical Configuration Audit (PCA)
Operations and Support	Initial Operational Test and Evaluation (IOT&E)

It is important to note that milestone rather than activity definitions are being used for the phases. Thus, for example, if some coding is done before CDR it is counted as part of the design phase. How the resources consumed will be separated by activity will be discussed in Sec. 2.4.

Secondly, the level of detail available changes with the phase. Starting with PDR and ending with PCA, developments are reported by Computer Program Configuration Items (CPCIs). By CDR the CPCIs have been further disaggregated into Computer Program Components (CPCs) and information by CPC is possible until PCA. The CPCs are themselves made up of modules of code, and information to this level is possible in the coding phase.

The proposed reporting system will gather information by phase and the potential level of detail will differ with each phase. The level of detail desired and the relationship of the resource consumption by phase to resource consumption by activity are specified subsequently.

2.2 SOFTWARE PRODUCT STATUS

There is no sense in reporting resource consumption to a level of detail greater than that describing the development status of the software product. Hence, we first defined data collection requirements for information on the software product, independent of the resources consumed.

These include the following for each CPC: Descriptive Information, Development Method, Milestone Information, Code Size Measurements, Code Changes, and Code Structure. Also requested is information on the software development as a whole: milestones, number of pages of specifications, number of test procedures, etc. Details are contained in Tables 8.1-8.3 of Vol. I.

Reporting requirements are related to phases, with single reports required at the milestones separating phases and recurring reports concerning code development status (baselining of source decks), changes to code during the qualification test phase, and code status during maintenance.

Milestone reports become more detailed as the development progresses, culminating in a very detailed description of the product at PCA. Recurring reports are more aggregated and do not require detailed documentation. For example, code status can simply be reported by delivering a copy of the source deck for each module, upon completion of coding and unit testing.

This continued reporting of source code completion will provide an almost continuous measure of product completion (e.g., 30% of the source code has now been written and debugged). Furthermore, actuals can be checked against estimates to see if man-hour consumption is in line with estimates, product size is within projections, and coding is on schedule.

We feel this innovation is the most important product status data item recommendation. To date no reliable data on product completion is available to the Program Office between CDR and PQT. This void of information has prevented the early recognition of problems and hence the implementation of early corrective measures. The requirement for source-deck delivery against a prearranged schedule will fill this void.

The milestone reports, while operationally not as meaningful, will provide a detailed and standardized statement of what was produced. The collection of reports will provide a history of the development. The entire set of data will provide the product description data base necessary for developing Cost Estimating Relationships.

2.3 SOFTWARE END ITEMS

A standard list of software end items has been selected from which we propose that Computer Program Components be selected (Table 2.1). We recognize that such a list is tentative and will undergo change with use. However, the need for a standardized list to disaggregate major defense system software into recognizable pieces is critical. Although each software development is unique, the parts are often common to many developments. Collection of data to a standardized list will eventually enable software cost to be estimated at the CPC level, much as hardware is now estimated on the component level.

2.4 DEFINITIONS OF REPORTING SYSTEM ELEMENTS

Software resources were divided into the following categories during the study. (We focused on contractor costs.)

Contractor

- Direct Technical and Support Labor
- Computer Resources
- Unique Facilities and Equipment
- Overhead
  - Support Labor
  - Materials
  - Travel
- G&A and Fee

Government

- Air Force Labor
  - Military
  - Civilian
- Technical Assistance Contractor Labor
- Government Furnished Equipment and Facilities

TABLE 2.1

SUGGESTED LIST OF COMPUTER PROGRAM COMPONENTS

(Table 8.4 of Volume 1)

1 SUPPORT SOFTWARE

1.1 EXECUTIVE

- 1.1.1 Computer Resource Manager
- 1.1.2 Computer/Peripheral Interface
- 1.1.3 Computer/Operator Interface
- 1.1.4 Computer/Computer Interface
- 1.1.5 Computer/Special Device Interface
- 1.1.6 System External Interface
- 1.1.7 System Failover and Recovery
- 1.1.8 Performance Monitoring

1.2 UTILITIES

1.3 LANGUAGE PROCESSORS

- 1.3.1 Assembler
- 1.3.2 Compiler
- 1.3.3 Interpreter
- 1.3.4 Translator

1.4 LOADERS

- 1.4.1 Bootstrap Loader
- 1.4.2 Linkage Editor/Loader

1.5 HARDWARE DIAGNOSTICS

1.6 SYSTEM SIMULATIONS, TESTS, AND EXERCISES

- 1.6.1 Operational Scenario
- 1.6.2 Control and Sequencing
- 1.6.3 Data Collection
- 1.6.4 Data Reduction and Analysis

1.7 SOFTWARE DEVELOPMENT AIDS

1.8 PROJECT MANAGEMENT AIDS

- 1.8.1 Schedule Maintenance
- 1.8.2 Financial Accounting

TABLE 2.1 (Continued)

2 APPLICATIONS SOFTWARE

2.1 AVIONICS SOFTWARE

2.1.1 Mission Planning

- Preplanned Mission Evaluation
- Real-Time Mission Modification
- Steering Coordination
- Weapon Delivery Coordination
- Waypoint Sequencing and Mode Selection

2.1.2 Navigation

- TACAN
- LORAN
- Doppler Radar
- Inertial Reference Unit
- Auxiliary Attitude Reference
- Air Data Computations
- Kalman Filter

2.1.3 Aircraft Steering

- Course Select
- Manual Course
- Instrument Landing System
- Airborne Instrument Landing & Approach
- Data Link
- TACAN

2.1.4 Flight Controls

- Roll, Pitch, and Yaw Controls
- Velocity/Acceleration Control
- Air Induction Control
- Energy Management and Control
- Cockpit Environment Control
- Crew Flight Input

2.1.5 Weapon Delivery

- Continuous Computation of Impact Point
- Weapon Miss Distance Computation
- Automatic Release Control
- Visual Release Control
- Stores Management and Control

TABLE 2.1 (Continued)

2.1.6 Sighting, Designation, and Fixtaking

- Forward Looking Radar
- Astro Tracker
- Laser Spot Seeker
- Low Light Level Television
- Forward Looking Infrared Radar
- Low Altitude Radar Altimeter
- Tracking Handle
- Electro-Optical Sighting
- Visual Sighting
- Fixtaking

2.1.7 Display Control

- Heads-Up Display
- Navigation Display
- Sensor Display
- Data Display

2.1.8 Data Entry/Retrieval

- Mission Entry
- Aircrew Panel Control
- Data Base Access

2.1.9 Communications

2.1.10 Electronic Countermeasures

- Threat Warning
- Electronic Warfare
- Penetration Aids

2.2 COMMUNICATIONS, COMMAND, CONTROL, AND INTELLIGENCE SOFTWARE

2.2.1 Data Acquisition

- Sensor Control
- Signal Processing
- Data Transfer

TABLE 2.1 (Continued)

2.2.2 Data Processing

- Mission Control
- Data Identification
- Data Reduction
- Data Manipulation

2.2.3 Operator Analysis and Decision Making

- Mission Management
- Data Display and Monitoring
- Operator Data-Entry and Control

2.2.4 Response Generation

- Communications Switching
- Message Processing
- Report Generation

2.2.5 Data Storage and Retrieval

3 DATA BASE

3.1 FILES

3.1.1 On-Line Updatable Files
3.1.2 Internal Files
3.1.3 System-Generated Files
3.1.4 Remote Data Base Files

3.2 INDEXES

3.3 TABLES

3.4 PROGRAM SUPPORT LIBRARY

3.4.1 Program Library
3.4.2 Macro Library

3.5 MAINTENANCE ROUTINES

Our recommended set of resource elements is listed in Table 2.2. Note that they are oriented to the activities required to develop the software rather than the milestone-defined life-cycle phases. The relationship of these resource elements to life-cycle phases is given in Table 2.3. It was extremely important to distinguish between the activity and milestone uses of these terms. While it is the allocation of resources to activities that is important to the efficient development of software, it is the milestone definitions that have often been used when disaggregated costs have been reported.

Note that the reporting levels as well as the elements themselves have been selected to avoid artificial allocations of resources. For example, the allocation of functions to CPCIs is included in Analysis and reported at the contract or system level rather than attempting an allocation to CPCIs. Also there is no attempt to separate redesign from recoding during Integration and Testing. These error-response activities are merely incorporated into the integration activity.

Also note that there is no provision for product improvement during Operations and Support. It is our recommendation that product improvement be reported as a separate development so that descriptions of the improvements can be determined and their costs identified. Only then can a cost/benefit determination be made.

The elements, phases, and reporting levels are then related to the contractor resources as shown in Table 2.4. The different types of resources (man-hours, computer hours, and costs) are identified and their reporting frequency and level specified.

TABLE 2.2

REPORTING SYSTEM ELEMENTS

(Table 8.6, Volume I)

Activity	Definition
CPC Level:	All man-hours for particular activities that can be identified to the CPC level
Design	Detailed CPC design
Coding	Coding and initial checkout of CPCs
Integration	Redesign and recoding in response to test error detection
Installation	Modification of code for site-specific application
Maintenance	Coding and integration in response to error reports (DRFs)
CPCI Level:	All resources for particular activities related to the CPCI and not assignable to CPs
Design	CPCI design, interfaces, and allocation of functions to CPCs and modules
Testing	Defining and carrying out CPCI-level software tests
Independent V&V	Independent verification and validation of CPCIs
Contract or System Level:	All resources for particular activities related to the contract or system and not assignable to CPCs or CPCIs
Analysis*	Studies to resolve conceptual problems and demonstrate capability to meet requirements, including algorithm development, allocation of functions to CPCIs, etc.
Testing*	Testing of software functions at system level, including hardware interface
Independent V&V	Independent verification and validation at system level
Management	
Engineering	All management personnel assigned to oversee the technical quality of the software product, including directing software development, directing the preparation and review of software portions of technical documents, preparation for and attendance at technical review meetings, on-site support during operations, fault isolation, etc.
Configuration	All management personnel assigned to oversee the maintenance of baselined software and software specifications and to document and incorporate approved changes to the baseline
Project*	Other management activities directly associated with the software product, including contract management, cost and schedule management, business and administration planning, directing and controlling the project, other than those technical and configuration control tasks identified above.
Training	Preparation of course material, demonstration and conduct of training courses if contracted, etc.
Documentation:	Writing, editing, publishing, reproduction, and dissemination of software documents required by the CDRL. Could be reported at CPCI level for more detail.
Technical	Part I and Part II Specifications, Test Plans, User's Manuals, specified technical reports, etc.
Configuration	Version descriptions, configuration index, change status reports, specification change notices, etc.
Program Management*	Software development plan, cost performance reports, program schedules, program milestones, etc.
Support*	Non-technical personnel totally assigned to the software portion of the contract such as key punchers, secretaries, etc.

*These items are sometimes difficult to separate from hardware.

TABLE 2.3

REPORTING SYSTEM ELEMENTS AND LIFE-CYCLE PHASES

(Table 8.14 of Volume I)

< LIFE-CYCLE PHASES >

Reporting System Elements	Analysis Contract Award to PDR	Design PDR to CDR	Coding & Checkout CDR to Source Baseline	Internal Test and Integration Source Baseline to PQT	Qualification Test PQT to PCA	Installation	Operations & Support
CPC Level							
Design	Authorized Early Design	Detailed Design, Flowchart	Authorized Late Design				
Coding	Authorized Early Coding	Authorized Early Coding	Coding and Debugging				
Integration				Error Response	Error Response		
Installation						Site-Specific Changes	
Maintenance							Priority Error Correction
CPCI Level							
Design	Authorized Early Design	Module/ Function Allocation, Interface Design					
Testing		Test Procedures		CPCI Tests	CPCI Tests		
Independent V&V					V&V Tests		
System Level							
Analysis	Special Studies	Authorized Late Analyses					Special Studies
Testing	Test Plan	System Test Procedures			System Tests	Contractor Off-Site Tests of Changes	Contractor Qualification Test of Error Correction
Independent V&V	Requirements, Validation	V&V Plan	V&V Procedures		V&V Tests	On-Site Qualification Test of Changes	Independent Qualification Testing of Error Correction
Management:							
Eng.	X	X	X	X	X	X	X
Config.		X	X	X	X	X	X
Project	X	X	X	X	X	X	X
Training		Training Plan		Training Manual Preparation		Initial User Training	Training and Demonstration
Documentation							
Technical	X	X	X	X	X	X	X
Config.		X	X	X	X	X	X
Pro.Mgmt	X	X	X	X	X	X	X
Support	X	X	X	X	X	X	X

TABLE 2.4

CONTRACTOR REPORTING OF RESOURCE CONSUMPTION

(Table 8.18 of Volume I)

In each category, estimated costs are to be reported once, at the beginning of the contract, except as noted. Actual costs are to be reported monthly, except as noted.

	Notes
Direct Labor (in dollars and in man-hours)	1, 2, 3
Computer Resources	
Computer Hours	1
Investment	4
Operations	
Unique Facilities and Equipment	
Investment	4, 5
Operations	6
Overhead Rate	
G&A and Fee Rates	

NOTES:

1. Disaggregated by computer program component as in Table 2.3
2. Estimates disaggregated by skill category as in Sec. 8.6.4 of Vol. I.
3. Estimates revised at PDR and at CDR.
4. Actuals reported when incurred.
5. Disaggregated by type of facility.
6. Disaggregated by type of facility or equipment.

Section 8 of Vol. I also specifies reporting requirements for unique facilities and equipment, computer system description (Table 8.16 of Vol. I) and utilization (Table 8.19 of Vol. I), and gives a standardized list of skill categories including job descriptions and current salary information (Appendix C of Vol. I).

2.5 FEASIBILITY OF IMPLEMENTATION

Specifying reporting system elements is only part of the requirement for a better software resource consumption data base. Implementation is the other (and perhaps most important) part. We therefore attempted, successfully, to match the reporting system elements to an extended version of the computerized reporting system defined in AFSCM 173-4.[3]

Without the extension, defined in Ref. 4, the Program Breakdown Structure of Ref. 3 is too aggregated for reporting software resource consumption. Briefly, the extension expands the software reporting code 42xx to accommodate a more detailed breakout.* First, the code is expanded to include interim standard Program Breakdown Codes (PBCs), which have the form

s421xxy

where
- s is a letter that identifies the software's supplier
- 421 identifies the element as a software product
- xx is an alphanumeric code that designates the software type
- y when used, is a letter (A, B, ..., F) that identifies the computer program life-cycle phase to which the element applies; if y is not used, the element is presumed to encompass all such phases covered by the contract

*This description reproduces the main parts of the description in Ref. 4.

In addition, the following software analogs for the standard WBS Level 2 categories are included:

PBC	Software Support Element Name
4210	Computer programs (coding, checkout, purchase, or rental costs only)
4220	Software-peculiar training
4240	Equipment required specifically for software development
4250	Testing of software
4260	Software-peculiar management and engineering
4270	Software documentation
4285	Software development and maintenance facilities
4200	Summary of all software-related costs

Furthermore, the extended code allows for the further designation of the product by providing three or more positions (after /) to designate segment, function, CPCI, etc. Thus, s421xxy/ABC designates resource consumption during phase y for CPC-xx, CPCI-C, of functional area B and Segment A.

We modified the y designator to refer to specific reporting elements rather than life-cycle phases. Life-cycle phase information can be recorded by summarizing the data at the milestones which separate the individual phases.

With this modification and the above extended system definitions, we assigned codes to each of the resource elements shown in Table 2.2. Results are reported in Tables 8.21 and 8.22 of Vol. I.

The ability to use this extended reporting system demonstrates that our recommendations can be implemented without the research costs and delays associated with data base software development.

In conclusion, we feel that implementation of this (or a similar) reporting system is of prime importance. We feel that we have integrated the three dimensions of the reporting system (life-cycle phases, end items, and resource consumption) into a workable and consistent set of reporting elements. Furthermore, we have incorporated reporting requirements for product description and status (Sec. 2.2) that will go a long way towards aiding the Program Offices in cost and product development control. We have tried to reach the right balance between the Government's needs for data and acceptance by contractors.

We are confident that if this reporting system is implemented, the Program Offices will have better project control and the software community will eventually have a good resource requirements data base.

3 RESOURCE ESTIMATING RELATIONSHIPS

As mentioned in Sec. 1.3, we were able to develop some new resource estimating relationships, despite the difficulties discussed in Sec. 1.2. The results address parts of the software resource estimating problem and do not form a complete model. Rather they are the development of some of the underlying relationships that precede the development of accurate cost models.

This section summarizes the results in four areas: (1) similarities between "ADP" and defense-system software, (2) aggregated relationships, (3) alternative product measures, and (4) maintenance activities. A detailed discussion of the development, applications, and equation derivations is contained in Sec. 6 of Vol. I. Our principal sources of data were the ADP Resource Estimating Procedures (ADPREP) study,[5] the Air Force Satellite Control Facility (SCF) Program Office, some GRC internal data, and some data from the open literature dealing with software reliability. The number of programs retrieved from PARMIS was too small to make it a statistically significant data source.

3.1 SIMILARITIES BETWEEN "ADP" AND DEFENSE-SYSTEM SOFTWARE

One of the first considerations is the reasonableness of using data on business-applications or "ADP" software to study and predict the costs of defense-system software.* In some phases, the cost of software development obviously depends upon the application.

Figure 3.1 illustrates how various types of software development relate to one another in terms of development effort for specified size. Generally, the defense system programs require more man-hours. However, commercial and defense-system programs exhibit similar trends, with somewhat more dispersion in the commercial data. We believe that the programming processes used to develop commercial and defense-system programs

*This is especially important with the reliance on PARMIS data for future studies recommended in this report.

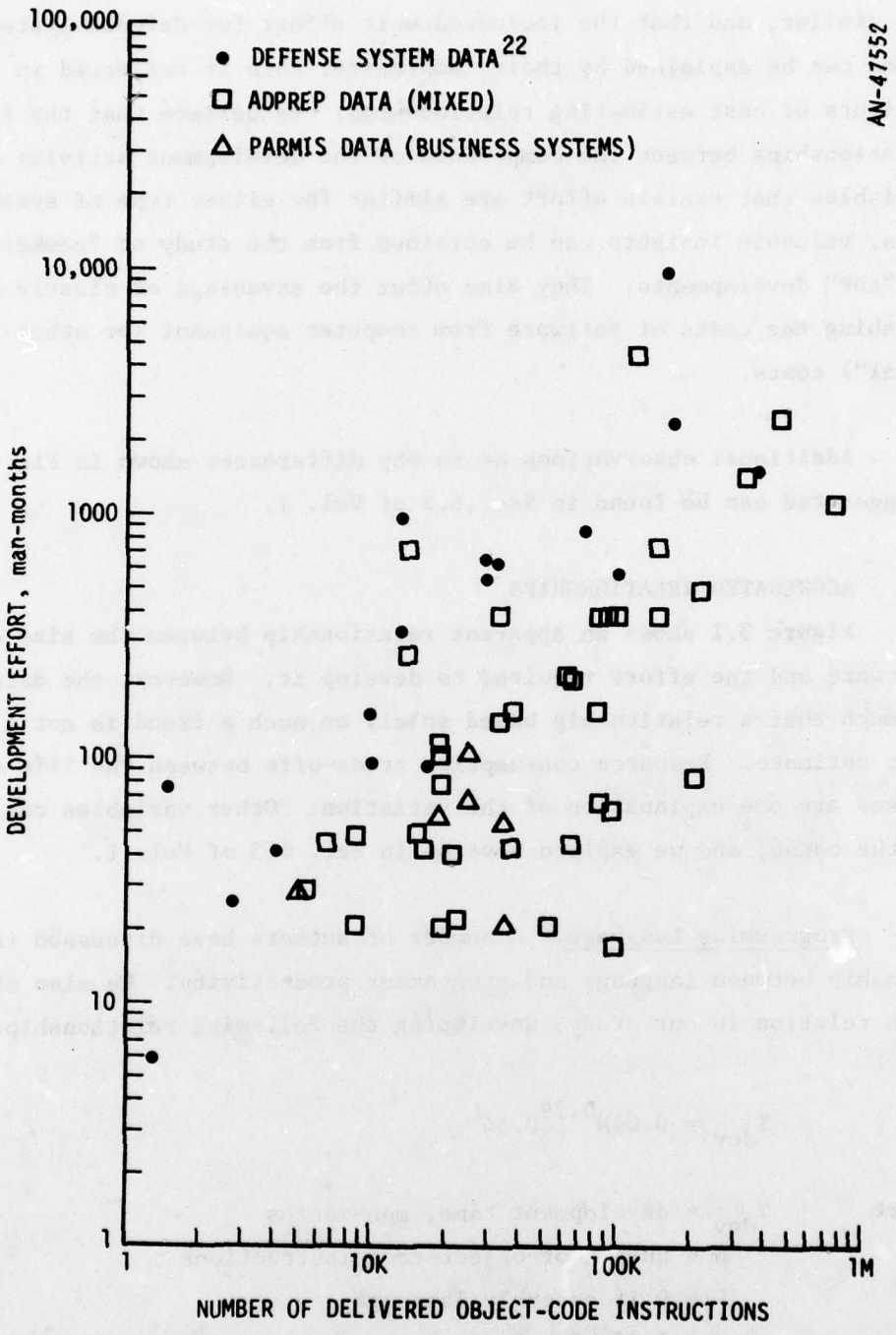

Figure 3.1. Size and Effort in Three Data Bases (Figure 6.1 of Vol. I)

are similar, and that the increased unit effort for defense-system programs can be explained by their complexity; this is reflected in the coefficients of cost estimating relationships. We believe that the functional relationships between the components of the development activity and the variables that explain effort are similar for either type of system. Thus, valuable insights can be obtained from the study of "commercial" or "ADP" developments. They also offer the advantage of clearly distinguishing the costs of software from computer equipment (or other "system level") costs.

Additional observations as to why differences shown in Fig. 3.1 are exaggerated can be found in Sec. 6.2 of Vol. I.

3.2 AGGREGATED RELATIONSHIPS

Figure 3.1 shows an apparent relationship between the size of the software and the effort required to develop it. However, the data varies so much that a relationship based solely on such a trend is not a useful cost estimate. Resource consumption trade-offs between the life-cycle phases are one explanation of the variation. Other variables can also be the cause, and we explore several in Sec. 6.3 of Vol. I.

Programming Language. A number of authors have discussed the relationship between language and programmer productivity. We also observed this relation in our study, developing the following relationship:

$$T_{dev} = 0.04N^{0.79}0.54^{L}$$

where T_{dev} = development time, man-months
N = number of object-code instructions
L = 0 if assembly language
= 1 if high-level language

As can be seen, the use of higher-order language is more productive, i.e., uses less man-months, than the use of assembly language. The quality of

this relationship is not high (low R^2),* but it was sufficient to establish the importance of programming language.

Hardware Constraint. In an earlier GRC effort,[6] we tested the effect of core-memory limitation on programmer productivity. The data had been allocated to three subsections (design, coding, and testing)** and equations were developed for each.

The results obtained are shown in Table 3.1. In each case, effort (man-years) is approximately linear with number of object instructions.

TABLE 3.1

HARDWARE-CONSTRAINT EQUATIONS

(Table 6.1 of Volume I)

$$T_{des} = 0.0030N^{1.03}5.64^{X}$$

$$T_{code} = 0.00049N^{1.13}8.33^{X}$$

$$T_{test} = 0.0080N^{0.934}5.15^{X}$$

where T_{des}, T_{code}, T_{test} = times for design, coding, and test (man-months)

$$X = \begin{cases} 0 & \text{if no hardware constraints} \\ 1 & \text{if more than 95\% of the core is utilized} \end{cases}$$

---

*R^2 is a statistical term commonly used to show goodness of model fit. The closer to 1, the better the fit. R^2 of 0.8 or above are good with 0.6 to 0.8 marginal.

**These breakouts are not equivalent to the phase definitions used in the current study.

Hardware constraints, when present, increase cost by a factor of five or more.

Design Changes. Experience with software developments has indicated that changes in requirements (through ECPs) can be an explanation of variations in product costs. Because of such changes, the delivered product size understates the size of the software which may actually have been written. Thus development effort per delivered instruction is considerably overstated.

Detailed data was not collected in sufficient quantity from PARMIS to analyze the direct impact of requirement changes on individual life-cycle phases. The ADPREP data[5] did report, by project, the number of changes made during development. Unfortunately, there was little discussion of the nature of the changes and no attempt to quantify the impact of an individual change. Thus, the only measure of change activity during development was the cumulative number of changes incorporated into the software.

In an attempt to use the available data, it was assumed that each change during development modified some average fraction of the software that was intended for delivery. The number of object-code instructions affected by a change was estimated as the product of the size of the delivered code, the cumulative number of changes during development, and the estimated fraction of the code that changed with each requirement change. It was hypothesized that effort was linearly related to delivered program size and to the amount of code that changed during development of the program. This was represented by an estimating equation that included the product of the number of requirements changes and the size of the delivered product, as well as the size of the delivered product:

$$T_{dev} = 6.61 + 0.000678N + 0.00011N \times C$$

where C = Number of changes (e.g., ECPs)

The R^2 for this relationship was 0.65, which is encouraging. The data does substantiate the cost of changes during development.

An application of a changing-requirements cost factor is to develop an initial cost estimating technique that requires the user to estimate the expected number of changes that will occur during the project's lifetime. In this way the impact of change can be anticipated and explicitly incorporated into the initial estimate of costs.

3.3. ALTERNATIVE PRODUCT MEASURES

Throughout the study, an attempt was made to relate measures of the product of a software development to the effort and costs associated with that product. The most obvious and commonly suggested measure is software size, expressed in terms of number of either source-language statements or object instructions. Each seems to have an appropriate application, and it is important that a consistent measure be used when comparing data from various projects. Inconsistency in this measure of program characteristics is probably, by itself, significant in explaining variation in the data reported, since it could vary by a factor of two to five.

Even if measures of computer program size can be converted to a normalized form (such as number of object instructions), another possible source of inconsistency exists. It has been argued that source-to-object-code relationships for high-level languages are invalid because differences in programming style (i.e., coding procedures and techniques) will result in significant variations from programmer to programmer.

The AOES data offered an opportunity to test this conjecture. The data base contains both the number of source-code card images for a large number of programs and the number of object-code instructions generated by a JOVIAL compiler. The entire set of JOVIAL programs which were considered ran on the same computer, so variations introduced by computer instruction sets are removed. Furthermore, the programs perform a wide range of

functions, from orbit determination to compilation and operating system functions, and are written by many authors. These AOES programs were then used to test the claim that significant variations in the relationship between the instruction counts of source-code and object-code are introduced by differences in programmer style. Regression analysis was used with the rationale that a weak linear relationship (i.e., low R^2) supports the hypothesis. The following equation was developed:

$$N = -30.495 + 2.4N_S$$

where, as before,

N = number of object-code instructions

N_S = number of source-code card images

A very strong linear relationship ($R^2 = 0.94$) was found between source and object code for this set of JOVIAL programs. These results indicate that the assertion is invalid; i.e., programmer stylistic variations in the use of JOVIAL are minor. Note that stylistic variations may still be an important determinant of number of source instructions.

This suggests that a table of conversion factors for language can be compiled for various programming languages and host computers. These factors could be applied to construct normalized measures of product size. Table 6.2 of Vol. I contains a compilation of such data based on the information used in this study.

Software development yields other products besides software, notably documentation. It is possible that a measure of documentation can serve as a surrogate for product size measured in source-language or object-language instructions. To test this conjecture, it was assumed that the size of the documentation describing a project would be related to the size of the final product, so long as common documentation requirements

and formats applied. Relaxing this last constraint would make any comparison meaningless.

The AOES data provides information that could be used to test this conjecture. For each of the many programs examined, the page counts for Part II Specifications were available, as well as measures of the product size in terms of the number of source statements. The following equation was developed:

$$N_S = 1600 N_P^{1.4}$$

where N_S = number of source-code instructions
N_P = number of pages of Part II Specifications

The relationship has a moderately low R^2 value (0.56) that indicates it should not be used to replace existing measures of software size. However, it does show promise and this type of relationship should be further studied. Such a relationship between documentation and program size could be used to check estimates of program size at the time of CDR when the first product-oriented documentation exists in draft form. If the relationship to documentation did not hold, the sizing of the program might then be subjected to further review, and a possible revision of project costs and schedule might be indicated.

This type of relationship also offers promise in directly calculating the cost of documentation. Since source cards are related to number of pages, an estimating equation for documentation could be calculated.

3.4 MAINTENANCE ACTIVITIES

Few previous cost estimating studies of software have examined this activity of the life cycle to any significant extent. In this study, it became apparent that the entire process was very poorly understood. The maintenance activity for software has two components: (1) error correction, and (2) improvement or refinement of the software product.

The AOES data provides some insight into the relative proportions of effort devoted to error correction and to product improvement. The data includes the allocation of direct technical man-hours during maintenance for the AOES project, in terms of the activities of the two associate contractors: one spent 25 percent of its man-hours on error correction and 75 percent on product improvement; for the other associate contractor, the division was 33 percent and 67 percent. Thus, for both contractors, most of the effort went to improvement rather than error correction.

The point is that a regular pattern of modification takes place and is to be expected during the operations period. However, these modifications are related to product improvement, rather than the initial development's success of failure in meeting original requirements. No trade-off exists between development and improvement, and a decision to fund the improvement should be based solely on costs and benefits of the improvement itself.

Resources consumed in software error correction, on the other hand, are clearly driven by the problems reported during operation of a system and the response times required to correct the reported errors. Further, these are and should be related to the software development effort.

The technical literature on software reliability indicates that the error process of operational software exhibits two major trends. First, the total number of problems encountered with a given model of software is limited, and the limit is related to the size of the software. Second, the mean time between error detections increases with use and corresponding error correction.

The data collected for the AOES project were at a sufficient level of detail to allow testing of some of these assertions. It was hypothesized that the number of problems reported, once a release of the computer program became operational, would be related to the size of the

computer program (residual errors), the new requirements introduced with the release of the modified computer programs (design changes), and the problems corrected within the computer program by the new release. The following relationship was derived:

$$\text{Number of Problems} = 0.4 + 0.00037N_S + 0.00069N_S \times N_C + 0.00048N_S \times N_P$$

where N_S = number of source-code instructions
N_C = number of design changes
N_P = number of problems corrected

The statistics indicate that each variable in the above equation is significant in explaining the reported problems with the software version once it was released for operational use. The R^2 (0.64) is marginal, but promising.

Finally, the choice or programming language should have a dramatic impact on maintenance. The goal of high-level programming languages is to make programs more readable and easily modified; therefore, programs in a high-level language should exhibit lower maintenance costs. The ADPREP data[5] contains information that allows this hypothesis to be tested. For each program reported on, the staffing of the maintenance function is described in terms of the total number of maintenance people assigned per month.

In Fig. 3.2, the number of maintenance personnel is plotted for programs written in assembly language (dots) and programs written in a high-level language (squares). Separate trends apparently exist, indicating that the use of a high-order language makes a significant difference in the staffing required for maintenance.

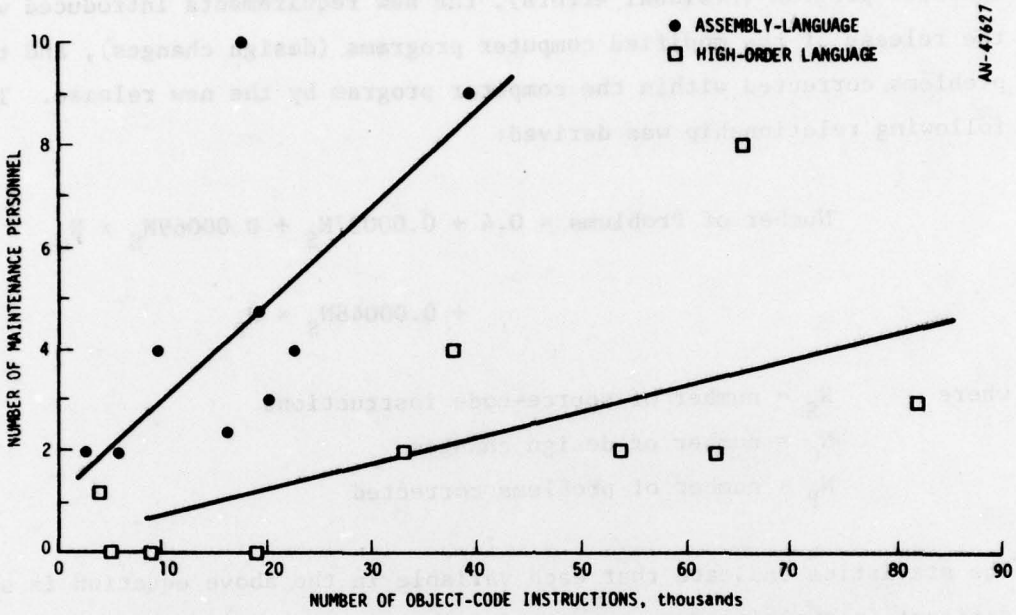

Figure 3.2. Maintenance Requirements for Assembly-Language and High-Order Language Programs (Figure 6.14 of Volume I)

A linear regression for the two languages was performed, with number of object-code instructions, N , as the independent variable. R^2 statistics are not very good, and the relationships are therefore tenuous.

The results were:

Assembly Language Programs

$$\text{Number of Staff} = 1.14 + 0.00019N$$

$$R^2 = 0.43$$

High-Order Language Programs

$$\text{Number of Staff} = 0.34 + 0.00005N$$

$$R^2 = 0.38$$

Based upon this initial examination of the maintenance activity, several conclusions about the maintenance process can be developed. The first is that the program-improvement aspect of maintenance is a regular, repeating process for computer programs that have a long and useful lifetime. This aspect of maintenance consumes a significant portion (if not the majority) of the software maintenance resources. How maintenance staffing is determined is not clear, but the extent to which programs are improved is probably based on the residual manpower available after error-correction activities for the currently operational version of the computer program.

Second, the error correction aspect of program maintenance appears to follow some general trends. The total number of problems one expects to encounter is probably limited, with the limit related to program size, the development history and the number of modifications installed in the program during maintenance. Furthermore, the mean time between software error detection/correction will increase with time and corresponding error correction. Thus, for a static program (without program improvements), one should expect to be able to taper the maintenance effort with time.

Third, the decision rule used to determine staffing levels is not clear although fewer resources are apparently required to maintain higher order language programs than assembly language programs. Maintenance staffing levels have apparently not been determined solely by estimates of anticipated error correction (true maintenance). Resources are included for product improvement (new development), although how and to what extent these determine staffing level is not clear.

REFERENCES

1. E.A. Nelson and T. Fleishman, A System for Collecting and Reporting Costs in Computer Program Development, System Development Corporation TM-3411/000/00, 11 April 1967.

2. Acquisition and Support Procedures for Computer Resources in Systems, AFR 800-14, 26 September 1975.

3. Cost Analysis: Program Breakdown Structure and Codes, Air Force Systems Command Manual 173-4, 24 November 1972.

4. J.B. Glore, Software Acquisition Management Guidebook: Statement of Work Preparation, Mitre Corporation, ESD TR 77-16, January 1977.

5. Automatic Data Processing Resource Estimating Procedures (ADPREP), Planning Research Corporation, PRC R-1527, August 1970.

6. E.N. Dodson, et al., Advanced Cost Estimating and Synthesis Techniques for Avionics, General Research Corporation, Final Report, CR-2-461, 1975.